UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/547,288	04/11/2000	Nir N. Shavit	1004-4663	4871

42714 7590 02/27/2007

SUN MICROSYSTEMS, INC.
ATTN: TIMOTHY SCHULTE
ONE STORAGETEK DRIVE, MS 4309
LOUISVILLE, CO 80028-4309

EXAMINER	
LI, AIMEE J	

ART UNIT	PAPER NUMBER
2183	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	02/27/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Office Action Summary	Application No.	Applicant(s)	
	09/547,288	SHAVIT ET AL.	
	Examiner	Art Unit	
	Aimee J. Li	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *07 August 2007 and 20 November 2006*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-43* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-43* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-43 have been considered. Claims 27, 28, 30, and 31 have been amended as per Applicant's request.

Claim Rejections - 35 USC § 101

2. 35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

3. Claims 40-42 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. Claim 40 recites "a computer program product encoded in at least one computer readable medium", which is further narrowed by claim 42 to include a network or other communication signals. As such, since Applicant felt it necessary to narrow the scope of "computer readable medium", which has not been defined in the specification to include signals, to "a disk, tape or other magnetic, optical or electronic storage medium and a network, or other communications medium", the Examiner assumes that the broader, independent claim 40 recitation of "computer readable medium" included these elements as well as others.

4. The Examiner would like to note that the previous rejection stated that the wires and "piece of paper" were examples of the non-statutory subject. The sentence "The wires, which are an embodiment of signals, are non-statutory" was meant to illustrate that not only the wires were non-statutory, but also any type of communications signal. The sentence about "instructions written on a piece of paper" was meant to illustrate that paper is a type of communication medium. The language "a network or other communications medium" still covers and heavily suggests communications signals, as well as transmissions, waves, links, fibers, pieces of paper with instructions written on them, etc., which are all non-statutory

materials. None of these elements are physical articles or objects and/or items which would structurally and functionally interconnect to the software in such a manner as to enable the software to act as a computer component and realize any functionality.

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

6. Claims 1-2, 4-5, 15-19, 25-28, and 30-31 are rejected under 35 U.S.C. 102(b) as being taught by Janice M. Stone's "A simple and correct shared-queue algorithm using Compare-and-Swap" ©1990 (herein referred to as Stone).

7. Referring to claim 1, Stone has taught a method of managing access to an array susceptible to concurrent operations on a sequence encoded therein, the method comprising:

a. Executing as part of a pop operation, an atomic dual target compare and swap (DCAS) to update a then-current, end identifying index for the array and an element of the array adjacent to that identified by the end identifying index (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. Returning from the DCAS, on failure thereof, an indication by which an empty state of the array is detectable (Stone Section **4. The Shared-Queue Algorithm** and Figure 5).

8. Referring to claim 2, Stone has taught wherein the indication by which the empty state of the array is detectable is indicative of presence of a distinguishing value in the adjacent element (Stone Section **4. The Shared-Queue Algorithm** and Figure 5).

9. Referring to claim 4, Stone has taught

a. Wherein the pop operation is a left pop operation (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

b. Wherein the end identifying index is a left-end index (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

c. Wherein the adjacent element is to the right of the identified element (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

10. Referring to claim 5, Stone has taught

a. Wherein the pop operation is a right pop operation (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

b. Wherein the end identifying index is a right-end index (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

c. Wherein the adjacent element is to the left of the identified element (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

11. Referring to claim 15, Stone has taught a method of managing concurrent access to double-ended queue (dequeue), the method comprising:

a. Employing, in an implementation of a pop operation, execution of a an atomic dual target compare and swap (DCAS) to interrogate instantaneous values of a first end index and a dequeue element adjacent to that identified thereby for a signature indicative of an empty state of the array, the signature including presence in that adjacent element of a distinguishing value (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),

b. Wherein successful execution of an opposing end pop operation includes execution of a DCAS to update a second end index and a dequeue element

adjacent to that identified thereby, the update of that adjacent element storing the distinguishing value therein (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

12. Referring to claim 16, Stone has taught wherein successful execution of a competing, same end pop operation includes execution of a DCAS to update the first end index and a dequeue element adjacent to that identified thereby, the update of that adjacent element storing the distinguishing value therein (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

13. Referring to claim 17, Stone has taught

a. Wherein the first end index is a left index and, if the state of the dequeue is non-empty, the dequeue element adjacent to that identified thereby is a left most element of the dequeue (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The**

Compare-and-Swap Instruction, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

b. Wherein the second end index is a right index and, if the state of the dequeue is non-empty, the dequeue element adjacent to that identified thereby is the right most element of the dequeue (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

14. Referring to claim 18, Stone has taught

a. Wherein the pop operation is a left pop operation and the opposing end pop operation is a right pop operation (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. Wherein the first end index is a left end index and the element adjacent to that identified thereby is adjacent to the right (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

15. Referring to claim 19, Stone has taught wherein the distinguishing value is encoded as a null value (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

16. Referring to claim 25, Stone has taught a method of managing concurrent access to an array susceptible to competing accesses at same and opposing ends thereof, the method comprising:

a. Executing as part of a first access operation, an atomic dual target compare and swap (DCAS) to update a first end identifying index and an element of the array corresponding to a then-current value thereof (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The**

A-B-A problem, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

b. Executing as part of a competing second access operation, a DCAS to update a second end identifying index and an element of the array corresponding to a then-current value thereof (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),

c. Wherein, if successful completion of one of the first and the second competing access operations results in a boundary condition state of the array, the DCAS of the other of the first and the second access operations fails and returns an indication thereof (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

17. Referring to claim 26, Stone has taught

a. Wherein the first access operation and the competing second access operation are competing pop operations (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

b. Wherein the array elements corresponding to the first and second indices are each adjacent to that identified by the respective index (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

c. Wherein the boundary condition state is an empty state (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The**

Shared-Queue Algorithm; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

d. Wherein the adjacent element referenced by the failing one of the competing pop operations encodes a distinguishing value signifying the empty state (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

18. Referring to claims 27 and 30, Stone has taught

a. Wherein the competing access operations are competing opposing end pop operations (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. Wherein the first index and the second index identify opposing ends of the array (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3;

Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

19. Referring to claims 28 and 31, Stone has taught

a. Wherein the competing access operations are competing opposing end pop operations (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. Wherein the first index and the second index identify same ends of the array (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

Claim Rejections - 35 USC § 103

20. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person

having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

21. Claim 3 is rejected under 35 U.S.C. 103(a) as being unpatentable over Janice M. Stone's "A simple and correct shared-queue algorithm using Compare-and-Swap" ©1990, as applied to claim 1 above, and further in view of Mark Allen Weiss's Data Structures and Algorithm Analysis in C++ Second Edition © 1999 (herein referred to as Weiss). Stone has taught the end identifying index and an opposing end identifying index delimiting the sequence (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; and Figure 5). Stone has not taught wherein the array encodes a double-ended queue as a circular buffer of bounded size. Weiss has taught wherein the array encodes a double-ended queue as a circular buffer of bounded size (Weiss page 111). A person of ordinary skill in the art would have recognized, and as taught by Weiss, a circular queue lets elements be added to the beginning of a queue after they have been enqueued, thereby allowing more elements to enter the queue even after all positions have been used at one point. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the circular queue of Weiss in Stone.

22. Claims 6-14, 23-24, and 32-43 are rejected under 35 U.S.C. 103(a) as being unpatentable over Janice M. Stone's "A simple and correct shared-queue algorithm using Compare-and-Swap" ©1990 in view of Mark Allen Weiss's Data Structures and Algorithm Analysis in C++ Second Edition © 1999 (herein referred to as Weiss).

23. Referring to claim 6, Stone has taught a method of managing access to an array susceptible to concurrent operations on a sequence encoded therein, the method comprising executing as part of a push operation, an atomic dual target compare and swap (DCAS) to update a then-current, end identifying index for the array and an element of the array identified by the

end identifying index (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5). Stone has not taught returning from an operation, on failure thereof, an indication by which a full state of the array is detectable. Weiss has taught returning from an operation, on failure thereof, an indication by which a full state of the array is detectable (Weiss pages 111-112 and Figure 3.61). A person or ordinary skill in the art at the time the invention was made, and as taught by Weiss, that checking whether the array is full or not is part of the push operation to ensure that a nonexistent location is not written to (Weiss page 111), thereby preventing a data memory location error from writing to an invalid array location. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to ensure a data memory location error does not occur.

24. Referring to claim 7, Stone in view of Weiss has taught wherein the indication by which the full state of the array is detectable is indicative of absence of a distinguishing value in the identified element (Weiss pages 111-112 and Figure 3.61).

25. Referring to claim 8, Stone in view of Weiss has taught

a. Wherein the push operation is a left push operation (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The**

Compare-and-Swap-Double Instruction, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. Wherein the end identifying index is a left-end index (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

26. Referring to claim 9, Stone in view of Weiss has taught

a. Wherein the pop operation is a right push operation (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. Wherein the end identifying index is a right-end index (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The**

Compare-and-Swap-Double Instruction, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

27. Referring to claim 10, Stone has taught a method of providing concurrent access to a data structure, the method comprising:

a. As part of an access to a first-end of the data structure, performing in alternate legs of a conditional branch:

i. A first atomic multi-way compare and swap on then-current contents of a first-end index store and a corresponding element of the double-ended data structure to disambiguate a retry state and a boundary condition state of the double-ended data structure (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

ii. A second atomic multi-way compare and swap on then-current contents of the first-end index store and a corresponding element of the double-ended data structure, the second multi-way compare and swap performing the access and, on failure thereof, returning an indication disambiguating a retry state and the boundary condition state of the double-ended data

structure (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),

b. Wherein the conditional branch discriminates between presence and absence of a distinguishing value in an element of the data structure corresponding to the then-current contents of the first-end index store (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

28. Stone has not taught wherein a double-ended data structure of bounded size implemented using a circular buffer technique. Weiss has taught a double-ended data structure of bounded size implemented using a circular buffer technique (Weiss page 111). A person of ordinary skill in the art would have recognized, and as taught by Weiss, a circular queue lets elements be added to the beginning of a queue after they have been enqueued, thereby allowing more elements to enter the queue even after all positions have been used at one point. Therefore, it would have

been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the circular queue of Weiss in Stone.

29. Referring to claim 11, Stone in view of Weiss has taught

a. Wherein the access includes a pop from the first-end of the double-ended data structure (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

b. Wherein the boundary condition state is an empty state of the double-ended data structure (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

c. Wherein the retry state results from a concurrently performed push or pop access at the first-end of the double-ended data structure (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double**

Instruction, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

30. Referring to claim 12, Stone in view of Weiss has taught

a. Wherein the access includes a push onto the first-end of the double-ended data structure (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

b. Wherein the boundary condition state is a full state of the double-ended data structure; and

c. Wherein the retry state results from a concurrently performed push or pop access at the first-end of the double-ended data structure (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

31. Referring to claim 13, Stone in view of Weiss has taught wherein the double-ended data structure includes a double-ended queue (dequeue) (Weiss page 111).

32. Referring to claim 14, Stone in view of Weiss has taught wherein the multi-way compare and swap is a dual target compare and swap (DCAS) (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

33. Referring to claim 23, Stone has taught a method of managing concurrent access to double-ended queue (dequeue), the method comprising:

a. Employing, in an implementation of a push operation, execution of an atomic dual target compare and swap (DCAS) to interrogate instantaneous values of a third end index and a dequeue element identified thereby for a signature of the dequeue, the signature including absence in that identified dequeue element of a distinguishing value (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),

b. Wherein successful execution of an opposing and push operation includes execution of a DCAS to update a fourth end index and a dequeue element identified thereby, the update of the identified dequeue element storing a value other than the distinguishing value therein (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

34. Stone has not taught the signature is indicative of a full state of the dequeue. Weiss has taught the signature is indicative of a full state of the dequeue (Weiss pages 111-112 and Figure 3.61). A person or ordinary skill in the art at the time the invention was made, and as taught by Weiss, that checking whether the array is full or not is part of the push operation to ensure that a nonexistent location is not written to (Weiss page 111), thereby preventing a data memory location error from writing to an invalid array location. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to ensure a data memory location error does not occur.

35. Referring to claim 24, Stone in view of Weiss has taught wherein successful execution of a competing, same end push operation includes execution of a DCAS to atomically update the first end index and a dequeue element identified thereby, the update of that adjacent element storing a value other than the distinguishing value therein (Stone Section **1. Introduction**;

Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

36. Referring to claim 32, Stone has taught a double-ended queue (dequeue) implementation comprising a computer readable encoding of at least a first access operation, execution of the first access operation operating at a particular end of the sequence and employing an atomic dual target compare and swap (DCAS) to update a corresponding one, but not both, of the left and right indices L and R and an element of the contiguous array adjacent to the contiguous array element identified thereby (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5). Stone has not taught

a. A contiguous array S of bounded size encoded in an addressable store;

b. A left index L and a right index R into the contiguous array, the contiguous array S, the left index L and the right index R together defining a circular buffer with state including a sequence of zero or more values encoded in the contiguous array between elements S[L] and S[R] thereof and wherein at least the S[L] and S[R] entries encode a distinguishing value.

37. Weiss has taught

a. A contiguous array S of bounded size encoded in an addressable store (Weiss page 111);

b. A left index L and a right index R into the contiguous array, the contiguous array S, the left index L and the right index R together defining a circular buffer with state including a sequence of zero or more values encoded in the contiguous array between elements S[L] and S[R] thereof and wherein at least the S[L] and S[R] entries encode a distinguishing value (Weiss page 111).

38. A person of ordinary skill in the art would have recognized, and as taught by Weiss, a circular queue lets elements be added to the beginning of a queue after they have been enqueued, thereby allowing more elements to enter the queue even after all positions have been used at one point. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the circular queue of Weiss in Stone.

39. Referring to claim 33, Stone in view of Weiss has taught

a. Wherein the first access operation includes a push (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. Wherein, on failure, the DCAS returns an indication by which a full state of the contiguous array is detected (Weiss pages 111-112 and Figure 3.61).

40. Referring to claim 34, Stone in view of Weiss has taught

a. Wherein the first access operation includes a pop (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. Wherein, on failure, the DCAS returns an indication by which an empty state of the contiguous array is detected (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

41. Referring to claim 35, Stone in view of Weiss has taught

a. Computer readable encodings of at least three additional access operations (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**,

paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),

b. Wherein the first and three additional access operations include push and pop operations at left and rights end of the sequence, respectively (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

42. Referring to claim 36, Stone has taught a concurrent shared object implementation comprising:

a. A computer readable encoding of push and pop operations defined to operate on elements of the contiguous array and on respective of the opposing indices (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),

b. Wherein the push operation employs a first instance of an atomic dual target compare and swap (DCAS) operation to update one of the opposing indices and a corresponding element of the contiguous array (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5), and

c. Wherein the pop operation employs a second instance of a DCAS operation to update one of the opposing indices and a corresponding element of the contiguous array while returning on failure, an indication by which an empty state of the contiguous array is detected (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

43. Stone has not taught

a. A contiguous array encoded in an addressable store;

b. Opposing indices into the contiguous array usable to delimit therebetween a portion of the contiguous array for storage of a sequence of zero or more data values; and

c. Returning on failure, an indication by which a full state of the contiguous array is detected.

44. Weiss has taught

a. A contiguous array encoded in an addressable store (Weiss pages 111-112 and Figure 3.61);

b. Opposing indices into the contiguous array usable to delimit therebetween a portion of the contiguous array for storage of a sequence of zero or more data values (Weiss pages 111-112 and Figure 3.61); and

c. Returning on failure, an indication by which a full state of the contiguous array is detected (Weiss pages 111-112 and Figure 3.61).

45. A person or ordinary skill in the art at the time the invention was made, and as taught by Weiss, that checking whether the array is full or not is part of the push operation to ensure that a nonexistent location is not written to (Weiss page 111), thereby preventing a data memory location error from writing to an invalid array location. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to ensure a data memory location error does not occur.

46. Referring to claim 37, Stone in view of Weiss has taught

a. Wherein concurrent shared object includes a dequeue (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3.**

Compare-and-Swap; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. Wherein the computer readable encoding of push and pop operations includes (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

c. Opposing end variants of the pop operation (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

d. Opposing end variants of the push operation (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The**

A-B-A problem, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

47. Referring to claim 38, Stone in view of Weiss has taught

a. Wherein concurrent shared object includes a queue or FIFO (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. Wherein the computer readable encoding of push and pop operations operate on opposing ends of the queue or FIFO (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

48. Referring to claim 39, Stone in view of Weiss has taught

a. Wherein concurrent shared object includes a stack or LIFO (Weiss page 93); and

b. Wherein the computer readable encoding of push and pop operations operate on a same end of the stack or LIFO (Weiss page 93).

49. Referring to claim 40, Stone has taught a computer program product encoded in at least one computer readable medium, the computer program product comprising:

a. Instances of the at least one functional sequence concurrently executable by plural processors of a multiprocessor and each including an atomic dual target compare and swap (DCAS) to update a corresponding one of the end identifying indices and an element of the array corresponding to a then-current value thereof (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. The DCAS of the at least one functional sequence responsive to a corresponding boundary condition state of the concurrent shared object (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

50. Stone has not taught at least one functional sequence implementing an access operation on a concurrent shared object, the concurrent shared object instantiable as a circular buffer of bounded size implementing a contiguous array delimited by a pair of end identifying indices. Weiss has taught at least one functional sequence implementing an access operation on a concurrent shared object, the concurrent shared object instantiable as a circular buffer of bounded size implementing a contiguous array delimited by a pair of end identifying indices (Weiss pages 111-112 and Figure 3.61). A person of ordinary skill in the art would have recognized, and as taught by Weiss, a circular queue lets elements be added to the beginning of a queue after they have been enqueued, thereby allowing more elements to enter the queue even after all positions have been used at one point. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the circular queue of Weiss in Stone.

51. Referring to claim 41, Stone in view of Weiss has taught

a. Wherein the at least one functional sequence includes opposing end variants of push and pop operations on the concurrent shared object (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

b. Wherein the boundary condition state corresponding to push operations is a full state of the array (Weiss pages 111-112 and Figure 3.61); and

c. Wherein the boundary condition state corresponding to pop operations is an empty state of the array (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

52. Referring to claim 42, Stone in view of Weiss has taught wherein the at least one computer readable medium is selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium and a network, wireline, wireless or other communications medium (Stone Abstract; Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

53. Referring to claim 43, Stone has taught an apparatus comprising:

a. Plural processors (Stone Abstract; Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double**

Instruction, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

b. A store addressable by each of the plural processors (Stone Abstract; Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

c. Means for coordinating competing access operations, the coordinating means employing in each instance thereof, at least one atomic dual target compare and swap (DCAS) operation to disambiguate a retry state and a boundary condition state of the array based on then-current contents of one, but not both, of first- and second-end index stores and an array element corresponding thereto (Stone Abstract; Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

54. Stone has not taught first- and second-end index stores accessible to each of the plural processors for identifying opposing ends of a bounded-size contiguous array encoded in circular buffer form in the addressable store. Weiss has taught first- and second-end index stores accessible to each of the plural processors for identifying opposing ends of a bounded-size contiguous array encoded in circular buffer form in the addressable store (Weiss pages 111-112 and Figure 3.61). A person of ordinary skill in the art would have recognized, and as taught by Weiss, a circular queue lets elements be added to the beginning of a queue after they have been enqueued, thereby allowing more elements to enter the queue even after all positions have been used at one point. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the circular queue of Weiss in Stone.

55. Claims 20-22 are rejected under 35 U.S.C. 103(a) as being unpatentable over Janice M. Stone's "A simple and correct shared-queue algorithm using Compare-and-Swap" ©1990, as applied to claim 15 above, and further in view of Mark Allen Weiss's Data Structures and Algorithm Analysis in C++ Second Edition © 1999 (herein referred to as Weiss).

56. Referring to claim 20, Stone has taught

a. Employing, in an implementation of a push operation, execution of an atomic dual target compare and swap (DCAS) to interrogate instantaneous values of a third end index and a dequeue element identified thereby for a signature of the dequeue, the signature including absence in that identified dequeue element of a distinguishing value (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**,

paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5),

b. Wherein successful execution of an opposing and push operation includes execution of a DCAS to update a fourth end index and a dequeue element identified thereby, the update of the identified dequeue element storing a value other than the distinguishing value therein (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

57. Stone has not taught the signature is indicative of a full state of the dequeue. Weiss has taught the signature is indicative of a full state of the dequeue (Weiss pages 111-112 and Figure 3.61). A person or ordinary skill in the art at the time the invention was made, and as taught by Weiss, that checking whether the array is full or not is part of the push operation to ensure that a nonexistent location is not written to (Weiss page 111), thereby preventing a data memory location error from writing to an invalid array location. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to ensure a data memory location error does not occur.

58. Referring to claim 21, Stone in view of Weiss has taught

a. Wherein the first end index and the third end index identify a same end of the dequeue (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. Wherein the second end index and the fourth end index identify a same end of the dequeue (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

59. Referring to claim 22, Stone in view of Weiss has taught

a. Wherein the first end index and the fourth end index identify a same end of the dequeue (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

b. Wherein the second end index and the third end index identify a same end of the dequeue (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

60. Claim 29 is rejected under 35 U.S.C. 103(a) as being unpatentable over Janice M. Stone's "A simple and correct shared-queue algorithm using Compare-and-Swap" ©1990, as applied to claim 25 above, and further in view of Mark Allen Weiss's Data Structures and Algorithm Analysis in C++ Second Edition © 1999 (herein referred to as Weiss). Stone has taught

a. Wherein the first access operation and the competing second access operation are competing push operations (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5);

b. Wherein the array elements corresponding-to the first and second indices are each identified by the respective index (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1**

The Compare-and-Swap Instruction, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5); and

c. Wherein the array element referenced by the failing one of the competing push operations encodes a value other than a distinguishing value (Stone Section **1. Introduction**; Section **2. Background**, paragraphs 3 and 5-6; Section **3. Compare-and-Swap**; Section **3.1 The Compare-and-Swap Instruction**, paragraph 1; Section **3.2 The A-B-A problem**, paragraphs 2-3; Section 3.3 **The Compare-and-Swap-Double Instruction**, paragraphs 1 and 4; Section **4. The Shared-Queue Algorithm**; Section **5. State diagram for the shared queue**, paragraphs 1-2 and 5; and Figure 5).

61. Stone has not taught wherein the boundary condition state is a full state. Weiss has taught wherein the boundary condition state is a full state (Weiss pages 111-112 and Figure 3.61). A person or ordinary skill in the art at the time the invention was made, and as taught by Weiss, that checking whether the array is full or not is part of the push operation to ensure that a nonexistent location is not written to (Weiss page 111), thereby preventing a data memory location error from writing to an invalid array location. Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to ensure a data memory location error does not occur.

Response to Arguments

62. The Examiner withdraws the objections to claims 27, 28, 30 and 31 in favor of the amended claims.

63. Applicant's arguments filed 07 August 2006 have been fully considered but they are not persuasive.

64. Applicant argues in essence on pages 12-13

> ...Applicant respectfully notes that claim 42 actually recites 'wireline, wireless' not wires. Furthermore, the Examiner does not identify anything within claim 40 to support the Examiner's conclusion regarding statutory subject matter...

65. This has not been found persuasive. While the claim has been amended eliminate the langue of wireline and wireless to overcome the rejection, the rejection is still upheld due to the presence of the language "a network or other communications medium". As further explained in the rejection above, the "wire" and "paper" references were provided as an example of the language in question and why. By Applicant's own amendments, the Examiner believes Applicant's were at least peripherally aware that communication signals, such as the waves found in wireless transmissions, are non-statutory subject matter with their voluntary amendment eliminating the "wireless" language from claim 42 but were unclear on what matter the Examiner considers falling under "a network and other communications medium" in their claim language. As for the reasoning behind the Examiner's rejection, please see MPEP 2601.01 for more information regarding non-statutory subject matter in computer-related applications. Also, the Examiner is unclear of the difference between "wire" and "wireline" as argued by Applicants. As such, the Examiner cannot formulate an appropriate response to the argument that there is a distinction between "wire" and "wireline", as suggested by Applicant's arguments.

66. Applicants argue in essence on pages 13 and 14-15

> An empty state is not detectable from an indication returned by the CSDBL...variously recite returning from a DCAS, on failure of the DCAS, an indication of an empty state when the DCAS is executed as part of a pop operation...

67. This has not been found persuasive. Applicant's arguments seen to be stating that the DCAS returns an empty or full state. However, that is not the claim language. The claim language is returning an "indication" of an empty state. Also, according to the specification, the DCAS itself does not return "empty" or "full". According to the code written specification for the push and pop operations, the DCAS is merely another level of checking whether to ensure the conclusion that the queue is empty or full is correct.

68. Before explaining Stone, the Examiner would first like to explain the purpose and functionality of the CSDBL. In section 3.2, Stone describes "The A-B-A Problem", which occurs when in shared queues when the item at the head of the queue has been dequeued, enqueued again, and is again at the head of the queue. Stone combats this problem with the CSDBL to maintain a counter with the queue pointer. As Stone states in section 3.2, paragraph 2 "The Compare-and-Swap-Double succeeds only when both the pointer and the counter have the values that were read earlier." This means that the CSDBL only succeeds, i.e. returns true, were no manipulations to the queue while the enqueue and/or dequeue were trying to operate, otherwise the CSDBL fails, i.e. returns false. Stone on page 497 explains that the CSDBL returns true or false depending upon whether CSDBL is successful or not. To summarize the CSDBL, the A1, A2, B1, B1, and C are passed as global parameters in the CSDBL function call.

When the concatenated value of A1 and A2 is equal to the double-size value C, then the concatenated value of B1 and B2 is stored in C and the condition-code is set to true. When the concatenated value of A1 and A2 is not equal to the double-size value C, then the left portion of C is set to A1, the right portion of C is set to A2, and the condition-code is set to false.

69. Stone teaches on page 500 a Dequeue procedure. When stepping through lines 10-30 on page 500, it can be seen that the condition code, cc, is set to the value returned by the CSDBL. Stone shows in line 17 on page 55, that, when the CSDBL function is called, the parameters are as follows:

A1 = PrivateT (which is the local copy of the tail pointer)

A2 = Tcount (which is the local copy of the Dequeue count)

B1 = 0 (which is equivalent to null)

B2 = Tcount

C = Qtail_and_Enqcount (which is the concatenation of the tail value and the enqueue count)

70. The CSDBL, based upon the page 497 and the summary above, then test if the concatenated value of PrivateT and Tcount is equal to Qtail_and_Enqcount. If they are equal, Qtail_and_Enqcount is set equal to the concatenation of 0 and Tcount and the condition-code is true. If they are not equal, Qtail_and_Enqcount is set equal to the concatenation of PrivateT and Tcount and the condition-code is false. Now, not only have values in the global parameters been manipulated and changed, but a value of true or false is set to the "cc" variable in the Dequeue procedure. Stone in line 18 tests if "cc" is true, then "xx" is set using the CSDBL. If "cc" is false, then "next" is tested for a 0. If "next" is 0, then the item is dequeued. If "next" is not 0,

then "cc" is set again according to the CSDBL function, but this time the global parameters are PrivateH (which is the local copy of the head pointer), Hcount (which is the dequeue count), next, Hcount+1, and Qhead_and_DeqCount (which is the concatenation of the head pointer and the dequeue count). Again, the CSDBL returns whether it was able to carry out the swap or not. In line 30, it can be seen that the only way to exit the repeat-until loop (lines 10-30) is with "cc" equaling true, meaning that the initial pointer and count values read at the beginning of the dequeue operation were correct. If the CSDBL did not succeed, that means the dequeue was operating with erroneous data and it must do the operations again, including checking if the queue is empty in line 13. Therefore, the CSDBL is an indicator of an empty queue when the initial run of the loop in lines 10-30 was erroneous and a following execution of the loop found the queue to be empty.

71. Applicants argue in essence on pages 14 and 15

> The IsFull function returns an indication of a full state as a Boolean value, not because of failure...the IsFull function is not a DCAS.

72. This has not been found persuasive. Applicant's arguments seen to be stating that the DCAS returns an empty or full state. However, that is not the claim language. The claim language is returning an "indication" of an empty state. Also, according to the specification, the DCAS itself does not return "empty" or "full". According to the code written specification for the push and pop operations, the DCAS is merely another level of checking whether to ensure the conclusion that the queue is empty or full is correct.

73. Stone teaches an enqueue operation on page 498, which repeats until the CSDBL succeeds, i.e. is true. Stone has not considered checking whether a queue is full, since he

assumes that the queue he is working with a queue of infinite length, which in turn means unlimited memory, not finite length and limited memory, as taught by Weiss. Weiss has taught that, when working with finite length arrays or queues, one way to overcome the length limitations is to make the queue circular, but the boundary conditions must always be monitored. That means, Weiss has taught the full state of a bounded queue must be monitored along with the empty state, which Stone has already taught monitoring. This means that, similar to the dequeue operation which is affected by the empty state of a queue, the enqueue operation must monitor the full state of a queue. To only check the full state at the outset of the enqueue procedure, e.g. before the repeat-until loop, would be erroneous, since the repeat-until loop executes the CSDBL is executed, which checks the ensure that the queue pointers and counters are correct. As with the dequeue procedure, the full state boundary condition must be checked should the CSDBL fail and the repeat-until loop must re-execute, since the correct pointers and counters could indicate a full state, which would make enqueuing impossible on a bounded queue.

74. Applicants argue on page 15 the differences between a "retry state and a boundary condition state". The Examiner believes the explanation above refutes this argument. It appears the Examiner that Applicant's arguments are trying to establish that there is no boundary condition checking within Stone's dequeue operation or Stone in view of Weiss's enqueue operation. As explained above, this is a mis-representation of the broad claim language being used. The claim language merely requires that the failure of the DCAS somehow indicate that queue is empty or full. As explained above, the CSDBL fails when erroneous pointer and counter data is used, so it indicates with its failure that the loop, which first tests the boundary

conditions, must re-execute. Should the boundary condition, i.e. empty or full state, be found true upon re-execution, which was indicated by the CSDB, the loop exits.

75. Applicant argues in essence on page 15 "Hence, Weiss fails to disclose or suggest encoding distinguishing values..." This has not been found persuasive. Weiss has shown that the pointers of his queue point to empty slots within the queue for illustration. However, empty slots contain some type of marker for the pointers to understand that the slot is empty. Otherwise, there would be no distinction between and empty or used slot. Also, Weiss's Figure 3.59 shows how to make an empty queue, which shows an explicit assignment of a value to the front and back. In addition, Weiss was taken in combination with Stone. Stone shows that an unused location is indicated by a null value.

Conclusion

76. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

77. A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

78. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Aimee J. Li whose telephone number is (571) 272-4169. The examiner can normally be reached on M-T 7:00am-4:30pm.

79. If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

80. Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

Aimee J. Li
20 February 2007

RICHARD L. ELLIS
PRIMARY EXAMINER